**Project report**

**DATA WAREHOUSE**

**Name: Moeez Ahmed Khan Section: DS-B**

**Roll no: 21i-1694**

**1. Project Overview**

In this section, you will introduce the context of the project, METRO store's need for a near-real-time Data Warehouse (DW), and the problem you are solving with the implementation.

This project focuses on the design, implementation, and analysis of a near-real-time Data Warehouse (DW) prototype for METRO, one of the largest superstore chains in Pakistan. The DW aims to facilitate online analysis of shopping behavior, providing valuable insights into customer transactions, product performance, and sales strategies. By implementing a near-real-time Extract, Transform, and Load (ETL) process using the MESHJOIN algorithm, the system enriches transactional data with master data from customers and products to help METRO optimize its selling strategies, such as product promotions, stock management, and sales forecasting.

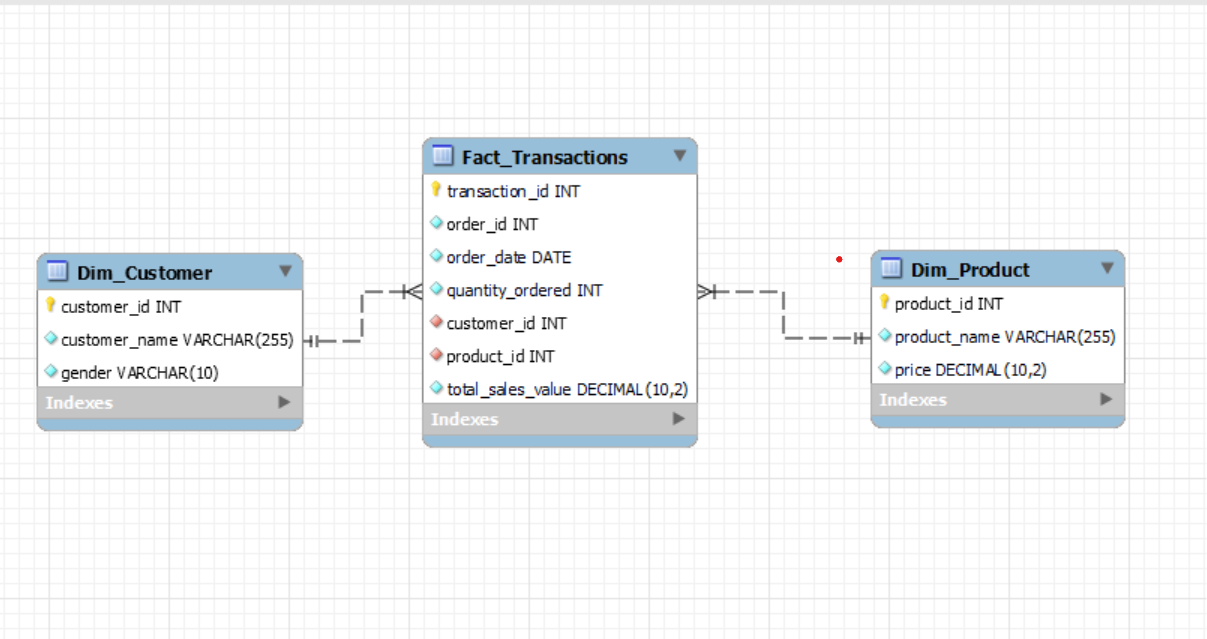
**2. Schema for DW**

In this section, describe the **star schema** you used for the Data Warehouse. Include a diagram (if necessary) showing the relationships between the fact table and the dimension tables, as well as a description of each table and its attributes.

The Data Warehouse (DW) schema follows the **star schema** design, which includes one fact table and multiple dimension tables.

* **Fact\_Transactions**: This is the central fact table that holds transactional data, such as ORDER\_ID, ORDER\_DATE, CUSTOMER\_ID, PRODUCT\_ID, QUANTITY, and TOTAL\_SALE.
* **Dim\_Customer**: This dimension table stores customer details, including CUSTOMER\_ID, CUSTOMER\_NAME, and GENDER.
* **Dim\_Product**: This dimension table stores product information such as PRODUCT\_ID, PRODUCT\_NAME, and PRODUCT\_PRICE.

The fact table is linked to the dimension tables via foreign keys (CUSTOMER\_ID in Dim\_Customer and PRODUCT\_ID in Dim\_Product).



**3. MESHJOIN Algorithm**

In this section, explain the **MESHJOIN** algorithm and how it works within the transformation phase of the ETL process. Describe the steps of the algorithm, how it handles streaming customer transactions, and how it joins data from the transaction stream with the master data.

The **MESHJOIN** algorithm, introduced by Polyzotis in 2008, is a stream-relation join operator designed to join streaming data with master data in an efficient manner. The algorithm processes customer transactions in chunks, loading the data into memory (hash table) and joining it with partitioned master data (e.g., customer and product tables). Each chunk of customer transaction data is processed cyclically with different partitions of master data, ensuring that all data is joined before it is loaded into the Data Warehouse.

The main components of the MESHJOIN algorithm include:

* **Disk Buffer**: A memory array used to load partitions of the master data.
* **Hash Table**: Stores customer transaction tuples and their join attributes.
* **Queue**: Keeps track of transaction chunks in memory, ensuring that all partitions of master data are joined with each chunk of customer transactions.
* **Stream Buffer**: Temporarily holds customer transaction data before joining.

The algorithm joins incoming customer transactions with the master data, calculates derived attributes (e.g., TOTAL\_SALE), and then loads the enriched data into the DW.

**4. Three Shortcomings in MESHJOIN**

In this section, identify three potential limitations of the **MESHJOIN** algorithm. Discuss why these limitations exist and what impact they may have on real-world applications.

While MESHJOIN is a powerful tool for stream-relation joins, there are some shortcomings:

* **Memory Usage**: The algorithm requires enough memory to hold both transaction chunks and the partitions of master data in memory. This can be an issue when the master data is very large or when there are limited memory resources available, potentially causing delays or memory overflows.
* **Latency**: Although MESHJOIN is designed for real-time processing, there can be latency when processing large amounts of data. The overlapping memory cycles and disk-buffer replacements might introduce delays in processing large transaction streams.
* **Scalability**: MESHJOIN assumes that the data can fit into partitions in memory, but when working with massive streams or large datasets, scalability could become a problem. The algorithm might struggle to scale efficiently across distributed systems or in environments where data size increases significantly.

**5. What Did You Learn from the Project?**

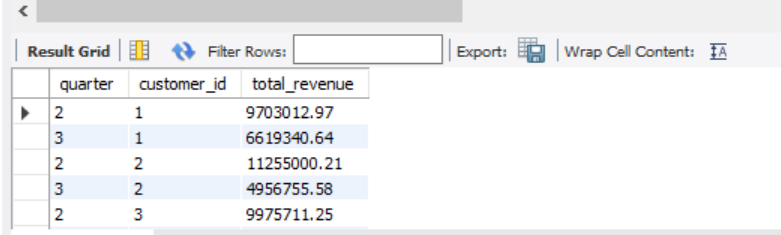
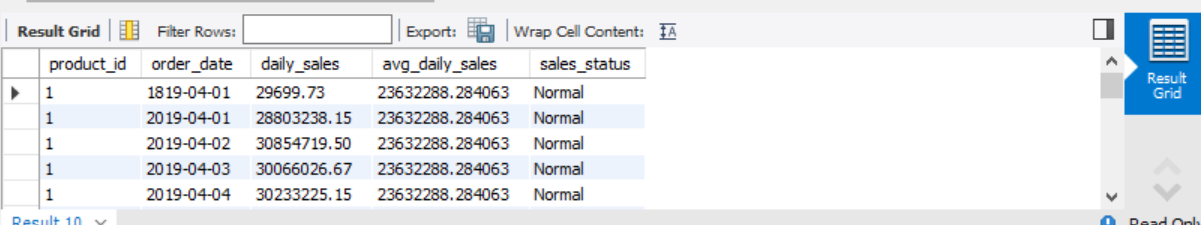
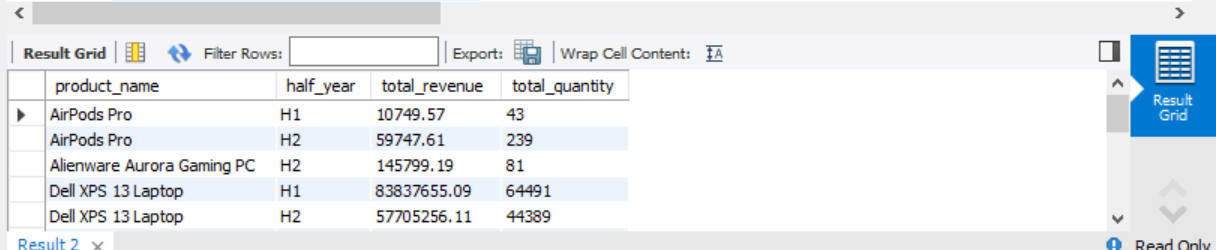
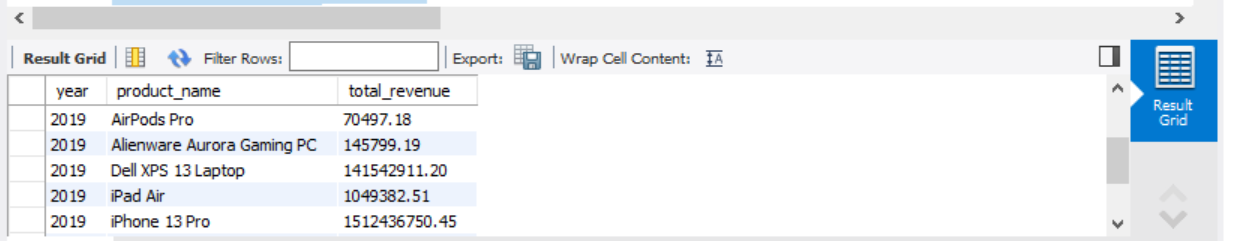
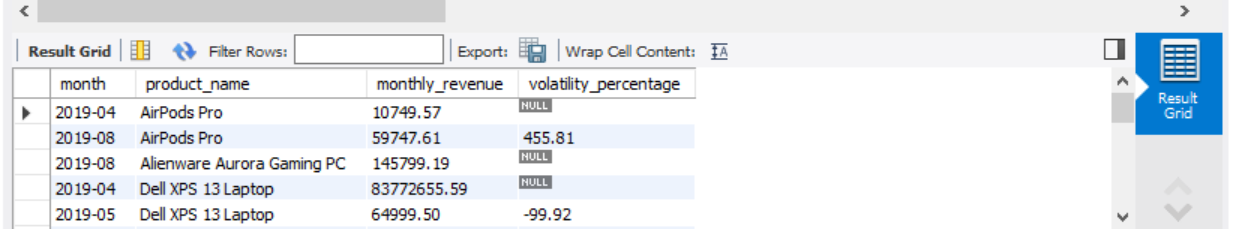
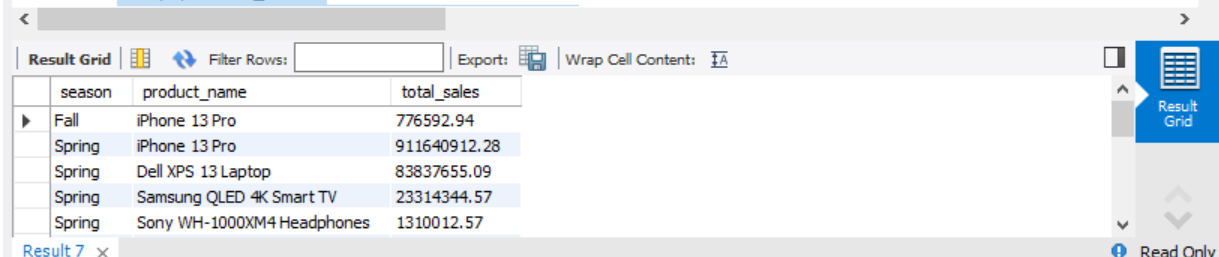
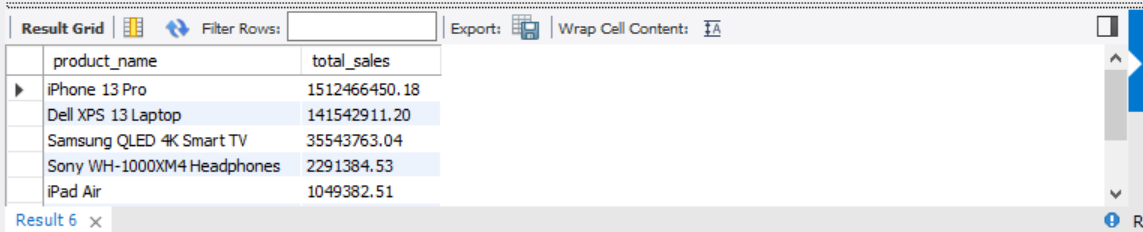
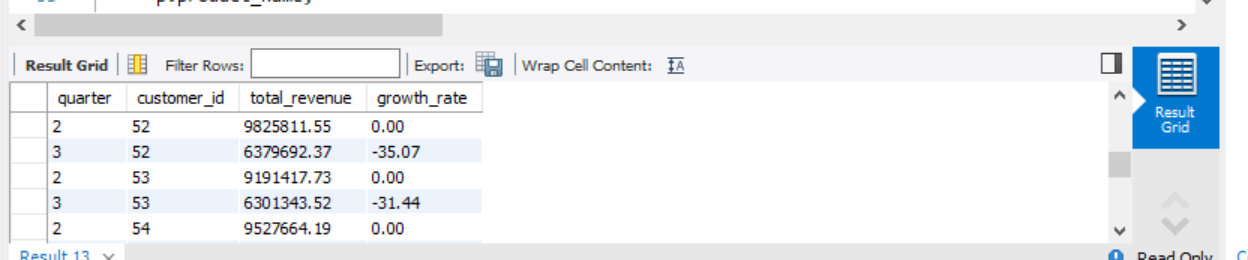
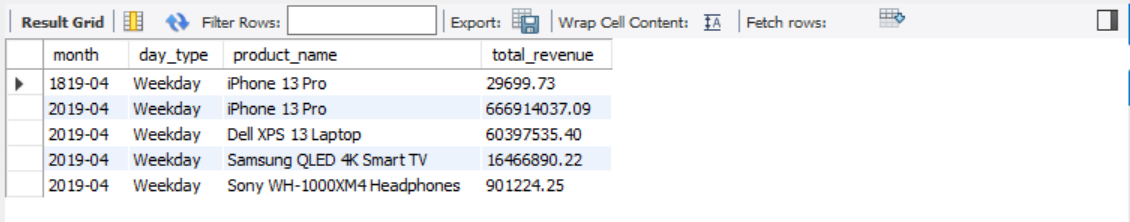
This section is where you reflect on the experience of completing the project, what you learned about Data Warehousing, MESHJOIN, ETL processes, and any challenges you faced along the way.

This project provided valuable hands-on experience in building a near-real-time Data Warehouse, specifically focusing on stream-relation joins in the transformation phase of ETL. I gained practical knowledge of the **MESHJOIN** algorithm and its application in enriching transactional data with master data, ensuring the seamless flow of information into the DW.

One of the key lessons I learned was the complexity of implementing real-time ETL pipelines and the challenges of handling large datasets. Despite these challenges, the **MESHJOIN** algorithm's ability to join data efficiently in a stream-processing context was impressive. Additionally, I learned the importance of optimizing memory usage and managing latency in real-time systems.

The OLAP queries helped me understand how multidimensional analysis can be performed on a DW, providing insights into business performance and customer behavior.

**Output of sql queries :**

****